ACEEE Int. J. on Information Technology, Vol. 3, No. 1, March 2013

A State-based Model for Runtime Resource Reservation for Component-based Applications

Ionut David, Rudolf H. Mak, and Johan J. Lukkien
Department of Mathematics and Computer Science, Eindhoven University of Technology,
P.O. Box 513, 5600 MB, Eindhoven, the Netherlands
i.david@tue.nl, r.h.mak@remove-this.tue.nl, j.j.lukkien@remove-this.tue.nl

***Abstract*— Predictable execution enforcement for applications with highly and arbitrarily fluctuating resource usage requires runtime resource management. Correct runtime predictions regarding resource usage of individual components allows making proper resource reservations, enabling a better resource management of the component-based applications. This work presents a state-based resource usage model for a component, in which states represent CPU utilization intervals. This resource model is intended for a resource-aware component framework where it will be used to determine the quality of resource reservation. For this purpose, the model offers two metrics: failure rate, which measures the fraction of the reservation periods for which the reserved budget was insufficient, and resource waste, which measures unused budget.To illustrate the model, we apply it to a family of reservation prediction strategies and validate the outcome by means of a series of experiments in which we measure the resource utilization of two video components. The latter requires a method for monitoring resource states which is also presented, analyzed and validated in this paper. [1]**

***Index Terms*— runtime resource management, component-based applications, resource model, resource usage, resource prediction**

I. Introduction

Component-based systems engineering is a software engineering technique that allows building complex systems from predefined building blocks. One of the main advantages of this technique is the possibility to distribute the components among different nodes. However, running component-based applications on distributed platforms implies also disadvantages as, for instance, difficulties in resource management. In this work, we are interested in the management of applications realized using a component-based software technology and exhibiting fluctuating resource demands. Examples of such applications are found in the area of video processing, both in multimedia [1] and in surveillance applications [2], where the required processing power is highly dependent on the video content.In previous work [3], David et al. (2010) sketched an architecture of a resource-aware component framework that targets such applications and identified a number of resource management strategies that can be applied in response to fluctuations in both resource demand and resource supply. These strategies range from local adaptations providing a remedy for demand-supply mismatches on a single node to system-wide reconfiguration that involves component migration, or even modification of the application topology through replication or replacement of components. In this paper, we address a true local strategy, namely dynamic resource reservation adjustment. In the context of multiple applications, resource reservations [4] are a means to manage the usage of shareable resources. In reservation-based resource management, each component of every application in need of a particular shareable resource is assigned a budget for that resource. Such a budget is called a reservation and it bounds the utilization of the resource by the component.

The typical goal of resource reservation adjustment is to make sure that resource provisions exceed resource demands, but only with a minimal amount. Making resource reservations and adjustments presupposes knowledge of component resource demands. When component demands are fairly stable and can be estimated at design time, appropriate resource reservations can be made at deployment time that last for the entire operation time of the application. Since these reservations must cater for the highest occurring demand, they are necessarily conservative. In systems with highly fluctuating resource demands, static resource reservation is therefore not suitable because it typically results in a too large waste of resources. From this we conclude that, for the class of applications we are interested in, reservations should be made at runtime and adjustment should take place on a regular basis. A potential downside is that this leads to 'best effort' behavior instead of 'guaranteed' behavior. Ideally, a component's resource reservation should be such that it resembles its resource utilization profile as closely as possible, but always being conservative. This means that resource reservation should be based on correct prediction of resource utilization.

Contributions. In this paper, we propose a resource model and reservation adjustment strategies that achieve this goal, in the following way. First, we introduce a state-based component resource model where resource utilization is classified in a number of so-called *states*. Second, we describe our monitoring method and analyze its accuracy. Third, we perform a number of experiments to validate the monitoring accuracy and to determine proper model (state size) and monitoring (sampling period) parameters. Finally, we examine a set of prediction strategies in which resource reservation is based on the prediction of the next utilization state from the current one. The main goal of this is to illustrate the effectiveness of the model which is expressed by two quality

[1]This research is funded partially from the ViCoMo project (ITEA2~08009).

DOI: 01.IJIT.3.1.1140

metrics for prediction: Fail, which measures how often an application lacks sufficient resources and Waste, which measures over-provisioning of resources. We limit attention in this work to the processor resource.

The organization of the paper is as follows. In Section 2 we discuss related work. In Section 3 we present the underlying resource model. In Section 4 we describe a series of experiments to validate model assumptions and to determine the effect of model parameters on the quality of the reservations for a simple prediction rule. In Section 5 we conclude with a summary of achieved results and directions for future research.

II. Related Work

In [5], existing approaches for model-based software performance prediction are classified into several categories: queueing network-based, process-algebra-based, Petri-nets-based, simulation-based and stochastic processes-based approaches. In this work we look at resource utilizations using a stochastic approach. In [6] and [7] the CPU resource usage is analyzed and modeled with the purpose of runtime adaptation and reconfiguration of the system. The work addresses mainly systems with architectural particularities regarding CPU usage (i.e. CPU throttling) and the focus is on the energy-awareness of the systems. The goal of our work is on achieving QoS-aware systems through runtime resource management.

In [8], [9] and [10] the resource consumptions of a particular class of medical video applications (i.e. interventional X-ray applications) are investigated and modeled with the goal of scalable and cost-efficient Quality of Service (QoS). The work demonstrates that when the resource usage of tasks is purely random or correlated for only short periods of time it can be predicted with Markov Decision Models (MDM). Another approach for prediction based on MDM and for a particular codec (i.e. MPEG-2) is found in [11], where the main goal is to maximize the quality of the output as perceived by the user by managing temporal load fluctuations. Our work addresses video applications in a more generic way and implies different runtime prediction strategies.

In [12] the resource consumptions of long-running applications are analyzed and modeled based on their history, with the goal of optimizing the runtime resource provisioning by means of observed resource usage patterns. The resource execution states are achieved by using a clustering algorithm. In our work, states are predefined.In [13], [14] and [15] a parameterized component resource model is presented, in which resource usage is captured by three values representing worst-case, best-case and average-case resource demand. These and other parameters are provided by the user of the framework and, once defined, are constant. In [10], a similar model is introduced for the description of component-based software architectures. The model is designed with a special focus on the design time prediction of quality attributes, e.g. performance and reliability.

In both [16], [17] and [18], the implicit assumption is that targeted applications are such that resource management can be treated as a design-time problem. It is also assumed that the designer will have a means to identify correct parameters.

Our component resource model targets applications characterized by highly and transiently fluctuating resource demands, e.g. video applications. Therefore, our resource model needs to accommodate resource management in which resource reservations are updated at runtime from predictions based on previous resource usage. In [19] a model for a specific video decoder component, viz. H.264, is shown that uses key metadata from the video stream to make resource predictions. However, resource prediction based on key metadata is not applicable to our case as we want a generic resource prediction that will be correct for any video processing component, or any component for that matter. While resource usage of a specific video decoding component can be related to key metadata, in the general case the resource usage of a component depends largely on the purpose of the component, the way it is implemented, the input data, and the internal state of the component.In [20] similar work for measuring the CPU utilization of video applications is shown, where resource monitoring is realized using a white-box approach, i.e., by code instrumentation of the components. In general, a white-box monitoring approach is not always possible.

Moreover, periodic sampling of multiple components on a single core will result in erroneous information in the recorded data. Even when all components succeed to sample their resource usage with the same sampling period, the measurements are by construction out of phase, since there can be only one process active on a core at any moment in time. In this work, we target periodic sampling and apply a black-box approach to simultaneously trace individual components of a component-based application by a separate monitor.

III. Resource Model

In this section, we present a resource model that addresses the situation in which a single component C uses a single shared resource R for an indefinite period starting at time t_s.

For $t_s \leq t_b \leq t_e$, let $u(t_b, t_e)$ denote the utilization of resource R by component C in the interval $[t_b, t_e)$, i.e., the fraction of R used by C. In this paper, we focus on CPU utilization, so the utilization is the fraction of the interval $[t_b, t_e)$ during which the CPU executes C. In general, the utilization depends on the input provided to C. Since we aim at a runtime model, the input to C is implicitly understood to be the current one, and therefore will not occur as a separate parameter in our model.Next, let $r(t_b, t_e)$ be the fraction of resource R reserved for component C in the interval $[t_b, t_e)$ Ideally, reservations should satisfy $u(t_b, t_e) \leq r(t_b, t_e)$ Reservation-based resource management for a component with a fluctuating resource demand involves the repeated determination of reservations r, each of which is valid for an

DOI: 01.IJIT.3.1. 1140

accompanying reservation period. For the sake of simplicity, we henceforth assume that these periods are equal and have duration P_{rsv} .So, for $0 \le i$

Let $t_i = t_s + i$. P_{rsv} be the moments at which these reservations are made, and let $u_i = u(t_i, t_{i+1})$ and $r_i = r(t_i, t_{i+1})$ be the utilization and reservation for the *i*-th interval $[t_i, t_{i+1})$, respectively. Collecting utilizations and reservations over *N* intervals results in a *utilization trajectory* $\mathcal{U}_N = \{u_i \mid 0 \le i < N\}$ and a *reservation trajectory* $\mathcal{R}_N = \{r_i \mid 0 \le i < N\}$ respectively.

We measure the suitability of a reservation trajectory for a utilization trajectory by two metrics. The first metric is the *failure rate*, defined as the fraction of the intervals for which the utilization exceeds the reservation, i.e.,

$$Fail(\mathcal{U}_N, \mathcal{R}_N) = \frac{1}{N} \#\{i \mid 0 \le i < N \wedge r_i < u_i\}, \quad (1)$$

where operator "#" denotes the cardinality of a set. The second metric is the *(resource) waste*, defined as the average reservation surplus, i.e.,

$$Waste(\mathcal{U}_N, \mathcal{R}_N) = \frac{1}{N} \sum_{0 \le i < N} \max(0, r_i - u_i). \quad (2)$$

Some remarks concerning these metrics are in place. First, a positive failure rate does not imply that an application crashes. In practice, it is more likely that it suffers occasional and transient reductions in QoS. Second, a positive waste does not imply that the CPU is idle during some reservation periods. An application may, and indeed should, release the CPU when it has consumed its necessary fraction u_i. Thus, it will give a scheduler the opportunity to allocate the remaining fraction $r_i - u_i$ of its reservation to other applications, e.g., those that execute without any reservation such as cycle scavengers. Third, any reservation strategy must involve a trade-off between these two metricssince they respond conversely to changes in reservation.A transition of a single reservation r_i from $r_i < u_i$ to $r_i \ge u_i$ leads to a decrease in failure rate by $1/N$, whereas the waste increases by $(r_i - u_i)/N$ due to the same change. Hence, we are also interested in the minimum waste $WM_n(\mathcal{U}_N)$ obtained by any reservation trajectory whose failure rate does not exceed the value $\frac{n}{N}$, defined as

$$WM_n(\mathcal{U}_N) = \min\left\{Waste(\mathcal{U}_N, \mathcal{R}_N) \middle| Fail(\mathcal{U}_N, \mathcal{R}_N) \le \frac{n}{N}\right\}. \quad (3)$$

In the sequel, we investigate reservation strategies in which reservation r_n is based on the preceding utilization trajectory $\mathcal{U}_n = \{u_i \mid 0 \le i < n\}$, which lend themselves to *run-time* reservation. For that purpose, we first limit the amount of information contained in a utilization trajectory by dividing the utilization range [0,1] into n_S equal-sized subranges, to which we shall refer as (utilization) *states*. To be precise, for any utilization value $u \in [0,1]$, its state $\sigma_S(u)$ is the rank of the utilization interval to which its belongs, i.e., we define $\sigma_S(0)=1$and $\sigma_S(u) = \lceil n_S u \rceil$, for $0 < u \le 1$ So $\sigma_S(u)$is a natural number satisfying $1 \le \sigma_S(u) \le n_S$, and for $u \ne 0$ we also have

$$(\sigma_S(u) - 1)S < u \le \sigma_S(u)S, \quad (4)$$

where S is the state size defined by $n_S S = 1$. In the sequel, we analyze the effect of *state-based reservation*, i.e., strategies whose reservations are an integral multiple of the state size *S*. Obviously, the state size influences the resulting waste. In particular, it is an immediate consequence of (4) that the minimum waste at zero failure rate is given by

$$MW_0(\mathcal{U}_N) = \frac{1}{N} \sum_{0 \le i < N} (\sigma_S(u_i)S - u_i) < S. \quad (5)$$

Recall that we target applications with highly and rapidly fluctuating resource utilization. Therefore, validity of (5) implicitly assumes that the reservation period is such that a reservation $r_n = \sigma_S(u_n)S$ is sufficient to cater for all fluctuations within the *n*-th period. It is also fairly obvious that the failure rate of state-based reservation is related to the accuracy of strategies to predict the next utilization state. If the predictions s_n of such a strategy satisfy

$$Prob(\sigma_S(u_n) > s_n) < \varepsilon,$$

for some predefined small value ε, then the expected failure rate is at most ε when we choose reservations accordingly, i.e., $r_n = s_n S$.

Because we want to predict the *n*-th utilization state u_n from the trajectory $\mathcal{U}_n$, we are interested in the nature of state transitions within these trajectories. In particular, reasonable predictions can be obtained, when, for any application input, state transitions between reservation periods are mostly between neighboring states.

So, for $1 \le a, b \le n_S$, let $\varphi(a,b)$, let denote the fraction of all state transitions in $\mathcal{U}_n$ that lead from *a* to *b*, i.e.,

$$\varphi(a,b) = \frac{\#\{0 \le i < N \mid \sigma_S(u_{i-1}) = a \wedge \sigma_S(u_i) = b\}}{N}, \quad (6)$$

where $\sigma_S(u_{-1}) = 1$ by definition. Moreover, for $-n_S < k < n_S$ let f_k denote the fraction of state transitions in which the resource utilization changes by more than kS i.e.,

$$f_k = \sum_{a+k<b} \varphi(a,b). \quad (7)$$

To illustrate the usefulness of utilization states, we analyze a class of very simple reservation strategies for CPU-usage. For $0 \le k < n_S$, we consider a strategy that produces the reservation trajectory $\mathcal{R}_N = \{r_i \mid 0 \le i < N\}$ with

$$r_i = min\left((\sigma_S(u_{i-1}) + k)S, 1\right). \quad (8)$$

In other words, the reservation for a period is based on the actual (monitored) utilization in the previous period and the "prediction" that the utilization state will increase by at most *k*. Of course, the state thus predicted should not exceed which means the reservation thus made should not exceed 1. In the sequel we will refer to *k* as the *prediction parameter* of the reservation strategy. Next, let $\mathcal{U}_N$ be a utilization trajectory and let $\mathcal{R}_N$ be the corresponding reservation trajectory generated by strategy *k*. It can be shown that the associated failure rate is given by

$$Fail(\mathcal{U}_N, \mathcal{R}_N) = f_k. \quad (9)$$

In addition, the waste can be bounded in terms of the failure rate and the state transition fractions. First note that, by (4), we have $u_i > (\sigma_S(u_i) - 1)S$ and, by (8), we have $r_i \le (\sigma_S(u_{i-1}) + k)S$ Application of these bounds to (2)

DOI: 01.IJIT.3.1.1140

gives the inequality

$$Waste(\mathcal{U}_N, \mathcal{R}_N) < W(k,S), \quad (10)$$

where the bound on the right hand side is given by

$$W(k,S) = \frac{1}{N}\sum_{0\le i<N} \max(0, (\sigma_S(u_{i-1}) + k - (\sigma_S(u_i) - 1)S)$$

Next, introducing names for states and eliminating terms with value zero, the right hand side of this inequality can be rewritten as

$$\sum_{a+k\ge b} \frac{1}{N} \sum_{\substack{0\le i<N \\ \sigma_S(u_{i-1})=a \wedge \sigma_S(u_i)=b}} (a + k - b + 1)S.$$

Since the terms of the inner summation do not depend on the index of summation, and expressing state transitions by summation instead of counting, i.e., using the equality

$$\varphi(a,b) = \frac{1}{N}\sum_{0\le i<N \wedge \sigma_S(u_{i-1})=a \wedge \sigma_S(u_i)=b} 1$$

instead of formula (6), we obtain

$$W(k,S) = \sum_{a+k\ge b} \varphi(a,b)\,(a + k - b + 1)S. \quad (11)$$

Obviously, the waste depends both on the state size *S* and on the prediction parameter *k*. Moreover, because the state transition fractions $\varphi(a,b)$ depend on the reservation period P_{rsv} ,so does the waste.

By a tedious but straightforward computation it can be shown that

$$W(k,S) = (1 - f_k)S + W(k-1,S). \quad (12)$$

This can be appreciated as follows.For strategy *k*, $(1 - f_k)$ is the fraction of reservation periods without failure, and therefore exhibiting some amount of waste. For those periods that would also be without failure at strategy *k-1* ,the additional waste is precisely *S*, for the others at most *S*

In Section 5, we present experiments that measure the influence of *k* on the failure rate and reservation waste. Furthermore, we investigate the robustness of the model in the context of a simple video application (i.e. a video player).

IV. Monitoring Method

In the previous section we have introduced a reservation strategy based on observed CPU usage in the past. An implementation of such a strategy requires the ability to monitor the CPU usage of an application, for which we must rely on operating system primitives with limited accuracy. Moreover, utilization, being defined as a relative quantity, can often not be measured directly by an OS primitive. Instead, the OS usually provides a pri-mitive to measure the amount of work done by the hardware (i.e. CPU) on behalf of a software component. In this section, we work out the details of such a (time-based) monitoring method for an operating system from the Microsoft Windows family. In particular, we consider a single application (containing one component) running on a dedicated CPU-core whose execution starts at time t_0. So, for $t \ge t_0$, let *w(t)* be the work (measured in time) spent by that core on *A* since t_0 Then, the utilization by of the CPU during the interval $[t_0, t)$ is given by

$$u(t) = w(t)/(t - t_0) \quad (13)$$

Next, assume that *u(t)* is sampled with a fixed period P_{smp} and let u_i be the measured utilization in period $[t_i, t_{i+1})$, where $t_i = t_0 + iP_{smp}$ Then

$$u_i = u(t_{i+1}) - u(t_i) = \frac{w(t_{i+1}) - w(t_i)}{P_{smp}} \quad (14)$$

The sampling period P_{smp} may, but need not, be the same as the reservation period P_{rsv} of the resource model.

In a Windows environment, an accurate measurement of the work is obtained using the Windows Performance Counters [21]. Given such a counter *PC* for *A*, we obtain an estimate $\overline{u}_i$ for u_i , given by

$$\overline{u}_i = (PC[i+1] - PC[i])/(\overline{t}_{i+1} - \overline{t}_i), \quad (15)$$

where $PC[i]$ is the value of *PC* inspected at time $\overline{t}_i$. In practice, the accuracy of $\overline{u}_i$ is determined by two factors:

i. the update granularity, i.e. the minimal amount by which the counter is incremented, and

ii. the inspection time jitter, i.e., the difference between the desired time of inspection t_i and actual time of inspection $\overline{t}_i$.

For the sake of simplicity we will assume that the effects of jitter are negligible, i.e., we will assume that $\overline{t}_i = t_i$ and consequently $\overline{t}_{i+1} - \overline{t}_i = P_{smp}$

Hence, it remains to determine the effect of the update granularity on the values recorded by the counter.

So, assume an update grain size . Then, at any time , the relation between the counter value and the actual work will be given by

$$PC(t) = \lfloor w(t)/T_u \rfloor T_u, \quad (16)$$

i.e., the value of the counter always contains the largest multiple of the update grain size smaller than or equal to the actual work performed by the CPU. So, for all $t \ge t_0$, we have

$$0 \le w(t) - PC(t) < T_u. \quad (17)$$

An immediate consequence of this counter granularity is that it imposes a lower bound on the state size *S* . Let n_C be the maximum number of counter updates during a sample period, then this is also the maximum number of states n_S that can be distinguished. Hence, we obtain

$$S = \frac{1}{n_S} \ge \frac{1}{n_C} = \frac{1}{\lfloor P_{smp}/T_u \rfloor} \ge \frac{T_u}{P_{smp}} \frac{\lfloor P_{smp}/T_u \rfloor}{\lfloor P_{smp}/T_u \rfloor}. \quad (18)$$

To all intents and purposes we replace the right hand side of (18) by the more conservative T_u/P_{smp} Next, we determine the maximum error in measured utilization, both per sample and in the average per trajectory. Using formulae (14) and (15) and recalling that we assume absence of jitter, we find for the error per sampling period

$$|\overline{u}_i - u_i| = \frac{|(PC(t_{i+1}) - PC(t_i)) - (w(t_{i+1}) - w(t_i))|}{P_{smp}}. \quad (19)$$

Combining (17) and (19) yields the upper bound

$$|\overline{u}_i - u_i| < \frac{2T_u}{P_{smp}} \le 2S. \quad (20)$$

Next, consider a complete trajectory. In the absence of jitter, and using $PC(t_0) = 0$, it follows from (15) that

$$\sum_{0\le i<N} \overline{u}_i = \sum_{0\le i<N} \frac{PC(t_{i+1}) - PC(t_i)}{P_{smp}} = \frac{PC(t_N)}{P_{smp}}$$

DOI: 01.IJIT.3.1.1140

Using $u(t_N) = w(t_N)/NP_{smp}$ and inequality (17) again, we obtain for the error in the average utilization per trajectory

$$\left|\left(\frac{1}{N}\sum_{0\le i<N}\overline{u}_i\right) - u(t_N)\right| = \left|\frac{PC(t_N)-w(t_N)}{NP_{smp}}\right| \le \frac{T_u}{NP_{smp}} \quad (21)$$

Observe, that the latter bound is inversely proportional both to *N*, the number of samples in the trajectory and NP_{smp}, in case of a video application, the total playing time . Of course, similar bounds hold for averages taken over arbitrary consecutive subtrajectories.

V. Experiments, Results, Discussion

In this section we describe the experiments that evaluate the state-based resource reservation strategies in the following way. First, we describe how to validate the monitoring method used for collecting the samples. Second, we identify the proper sampling period for monitoring. Third, we describe how to determine an appropriate reservation period for conducting these experiments.

The software component used in the experiments is a simple video player, based on the OpenCV library [21]. This OpenCV-based video player executes two types of activities: decoding a video stream and displaying it. These activities are performed by the component on a frame by frame basis, yielding an alternating sequence of decoding and displaying steps. Furthermore, during the experiments, we enable a video filtering feature of the video component. This increases the amount of work and thereby yields a considerable number of occupied utilization states.

The hardware platform used in the experiments comprises a desktop PC equipped with an Intel® Core™ i7 860 CPU model [22], running at 2.80 GHz. The operating system is Windows 7. This operating system allows monitoring of CPU usage through performance counters that are updated with a fixed granularity of T_u = *15.6 ms* , which means that the counter is increased whenever the CPU has spent another amount T_u of time on the associated process. To avoid inspection time jitter, we execute the resource monitor that records u_N on a separate core, using busy waiting and assigning real-time priority to prevent other applications from suspending the monitor beyond the next sample time, as could happen when using a sleep(t) primitive to let the monitor idle until sampling is due.

A. Monitor Error Estimation

Consider a set of $\left\{\overline{u}_N^{(j)} \middle| 1 \le j < M\right\}$ of *M* runs on the same input, and let $\Delta_M(\overline{u}_N)$ denote the difference between the maximum and the minimum of measured utilization averages $\left(\sum_{0\le i<N}\overline{u}_i^{(j)}\right)/N$ in this set.

Furthermore, let the error coefficient *E* be defined by.

$$\Delta_M(\overline{u}_N) = E \cdot (T_u/NP_{smp})$$

Then, according to (21), for any two runs of the application on the same input the measured average CPU-utilization should differ by at most $2T_u/NP_{smp}$

Hence, in theory, should stay below 2, but in practice there will be some deviation due to model abstractions and the fact that the work per run may vary slightly. Table I presents the outcome of an experiment of runs with fixed playing time, but varying sampling periods. It can be observed that in all cases the error coefficient is as expected. Table II shows the behavior of the error coefficient when the playing time increases. Again the error coefficient is almost constant, showing that the accuracy of the measured average utilization indeed scales inversely proportional to the playing time. The results of the experiments presented in this section imply that the monitoring method is valid and has a bounded error behavior.

Table I. Error Estimation With Fixed Playing Time But Varying Sampling Periods

N	P_{smp}	$\frac{P_{smp}}{T_u}$	$\frac{T_u}{NP_{smp}}$	$\Delta_M(\overline{u}_N)$	E
2500	156 ms	10	0.4 e-4	1.28e-4	3.2
1250	312 ms	20	0.4 e-4	0.76 e-4	1.9
500	780 ms	50	0.4 e-4	0.38 e-4	1.0
250	1560 ms	100	0.4 e-4	0.38 e-4	1.0
125	3120 ms	200	0.4 e-4	0.76 e-4	1.9
50	7800 ms	500	0.4 e-4	0.38 e-4	1.0

Table II. Error Estimation With Varying Playing Time And Varying Sampling Periods

N	P_{smp}	$\frac{P_{smp}}{T_u}$	NP_{smp}	$\frac{T_u}{NP_{smp}}$	$\Delta_M(\overline{u}_N)$	E
25	1560 ms	100	39 s	0.4 e-3	2.63 e-4	0.7
50	780 ms	50	39 s	0.4 e-3	2.63 e-4	0.7
100	1560 ms	100	156 s	0.1 e-3	1.78 e-4	1.8
250	1560 ms	100	390 s	0.4 e-4	0.84 e-4	2.1
1000	1560 ms	100	1560 s	0.1 e-4	0.29 e-4	2.9
1250	3120 ms	200	3900 s	0.4 e-5	0.08 e-4	2.0
2500	1560 ms	100	3900 s	0.4 e-5	0.09 e-4	2.3

B. Determination of Sampling Period

The goal in this section is to establish the appropriate period P_{smp} which will be used for monitoring the components. In order to do this, we perform an experiment that consists of sampling the resource usage of a video component with different values of P_{smp} for a sufficiently long playing time (65 minutes) and movie length (97500 frames at 25fps frame rate). The movie used in the experiment was *xvid* encoded and placed in an *avi* container.The results of the experiment are displayed using histograms. Figure 1, displays the utilization state frequency profile of a measured trajectory. On the X-axis the CPU utilization is represented, using a state size of *S*=0.05, resulting in 20 states. On the Y-axis the occurrence frequency (in %) of a state in the trajectory is given. As can be observed by the gaps between the occupied states, a sampling period of 156*ms* is too small. This is consistent with formula (14) which tells us that at this sampling rate only 10 states can be discerned. As the sampling period is increased, the profile converges to a profile consisting of a single occupied state that represents the average utilization over the entire trajectory. As can be seen, convergence is almost achieved at a sampling period of 1560ms, which at the given frame rate of 25fps corresponds to processing 39 frames.

DOI: 01.IJIT.3.1.1140

Since our goal is to provide accurate runtime management of resources by capturing rapid state fluctuations, however, we should aim at a sampling period where the resource usage profile is not dominated by a single state. Therefore, the experiment suggests that a sampling period P_{smp} around 500*ms* is appropriate.

Furthermore, it does not make sense to have a reser-vation period P_{rsv} smaller than the period at which we monitor the utilization. Combination of this observation with formula (14) yields the inequality $n_s T_u \leq P_{smp} \leq P_{rsv}$.

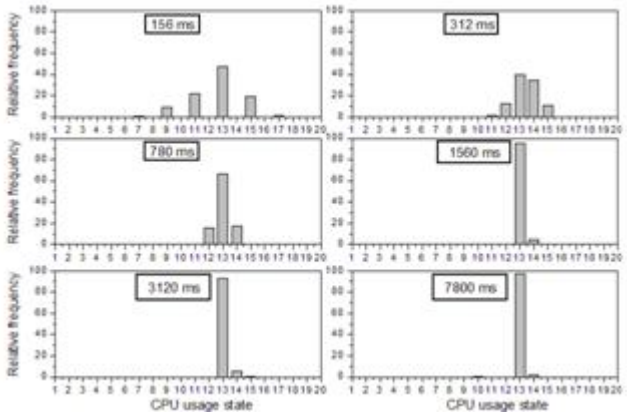

Fig. 1. The utilization state frequency profile of the CPU usage trajectory of OpenCV-based video player, monitored with P_{smp} Î {156ms, 312ms, 780ms, 1560ms, 3120ms, 7800ms}

C. Prediction Strategy Evaluation

In this section the resource reservation strategies defined in (7) are evaluated by means of the two metrics introduced in Section 3: failure rate *Fail* (1) and resource waste *Waste* (2). Since these metrics depend both on the prediction parameter *k* and the state size *S* , this leads to an exploration of the (*k*,*S*) space. From (7) it follows that for any fixed state size *S* the failure rate is decreasing with *k*, i.e., $f_{k+1} \leq f_k$.

Similarly, it follows from (2) and (8) that the resource waste is increasing with *k*. Keeping *k* fixed and increasing the state size *S*, it can be expected that a similar relationship exists, because making reservations in larger chunks tends to overshoot the required resources more and more often, thereby increasing waste and reducing failure rate. Nevertheless, there is no mathematical certainty.

To begin with, the resource usage state transition frequency profile of the monitored component, when the sampling period was set to *Psmp* = 780*ms* and state size *S* = 0.02, is visualized in Figure 2.

On the horizontal axes of the graph the previous (a) and the current (b) resource usage state of the monitored component are represented. On the vertical axis, the state transition frequency function φ is plotted. Observe, that all states are in the range [25..36] (of a total of 50 states) and that most of the state transitions are clustered around the main diagonal *a*=*b*. This indicates a stable resource usage behavior, where transitions are mostly between neighboring states. A more quantitative illustration of this fact is given in Table III, where the values f_k are used to compute the fractions

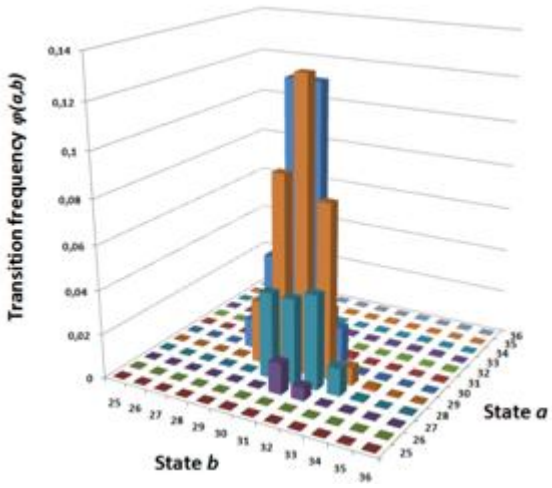

Fig. 2. State transition frequency profile of a video player; P_{smp} = 780*ms*; N=5000; *S* = 0.02

the state transitions that show an increase of more than *k* states (top row), a decrease of more than *k* states (middle row), or at most *k* states in either direction (bottom row). Observe that the entries in the latter row rapidly approach 1.

Next, we computed the quality metrics *Fail* and *Waste*, in order to estimate the quality of the resource prediction method introduced in this work. For that purpose, we applied a series of strategies *(k,S)*, with reservation parameter *k* varying over the set and usage state size *S* varying over the set $\{0.02, 0.04, 0.05, 0.1\}$ to the same utilization trajectory that gave rise to Fig. 2. The outcome of that computation is shown in Table IV. In addition, we determined the minimum waste *MW* obtainable for each strategy, i.e., the waste given by (3), where we took $n = Nf_k$ For the sake of clarity, all quantities in Table IV are given as a percentage rather than as a fraction. For f_k and W_k we indeed observe the monotonicity patterns described at the beginning of this section. Moreover, note that for fixed *S* the actual waste behaves almost identical to its upper bound as given by (12).

Finally, note that grey shaded entries represent strategies that are dominated by other ones. In this case, they are all dominated by the entry *(k,s)* = (4,4) which happens to have the minimal failure percentage of 0.02 % (the reservation for the first period is always too small). In general, strategies can be dominated by ones that use a smaller state size as illustrated by the next example. In this example, we consider resource prediction of a *motion detection* component applied with the same reservation period, and applied to the same video, as before. For this component the state transition frequency profile is given in Fig. 3

TABLE III. STATE TRANSITION FRACTIONS

k	0	1	2	3	4	5	6
f_k	0.3966	0.2064	0.0880	0.0220	0.0018	0.0008	0.0004
$1-f_{k-1}$	0.3880	0.2112	0.0900	0.0222	0.0026	0.0004	0.0000
$f_{k-1}-f_k$	0.2154	0.5824	0.8220	0.9558	0.9956	0.9988	0.9996

DOI: 01.IJIT.3.1. 1140

TABLE IV. FAIL AND WASTE FOR VARIOUS STRATEGIES – VIDEO PLAYER; P_{SMP} = 780MS; N=5000

k	S(%)	2	4	5	10
0	f_k (%)	39.66	26.54	26.52	17.22
	W_k (%)	2.64	3.44	3.60	4.74
	MW(%)	1.21	1.96	1.82	3.01
1	f_k (%)	20.64	6.56	6.34	0.40
	W_k (%)	4.22	7.05	8.06	14.68
	MW(%)	1.59	2.76	2.71	4.68
2	f_k (%)	8.80	0.16	0.06	0.02
	W_k (%)	6.05	11.02	13.03	24.68
	MW(%)	1.82	3.02	3.03	4.68
4	f_k (%)	0.18	0.02	0.02	0.02
	W_k (%)	10.00	19.02	23.02	42.90
	MW(%)	2.00	3.02	3.03	4.68

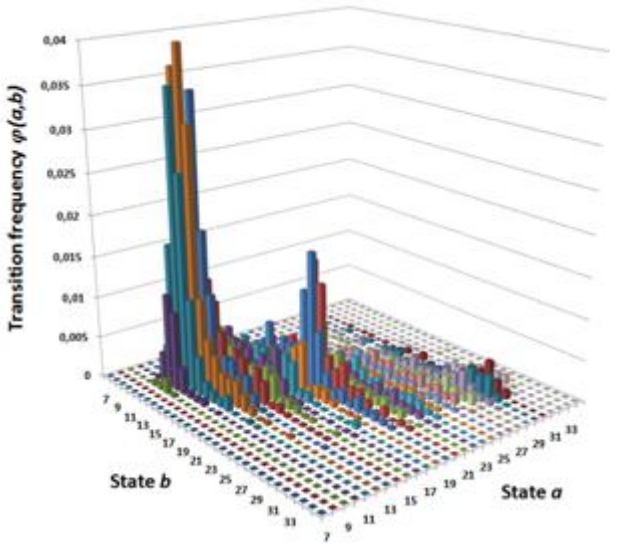

Fig. 3. State transition frequency profile of a motion detection component; P_{smp} = 780ms; N=5000; S = 0.02

Note that utilization of this component covers far more states than the one in Fig. 2. Nevertheless, state transitions are once again clustered along the main diagonal. In addition, we see three peaks indicating clusters of densely populated states.The Waste and Fail statistics of the various strategies for the motion detection component are given in Table V. Besides the monotonicity properties already described before, we now also see more interesting dominance patterns. Strategy *(k,s)* = (0,5) is dominated, i.e. lower failure rate and lower waste, by (1,2) and (1,5) is dominated by (0,3). All other points are Pareto points, showing that, in general, there will be a trade-off between resource waste and failure rate, depending on the QoS desired for the application.

CONCLUSIONS AND FUTURE RESEARCH

This work introduces a state-based resource model suitable for making, at runtime, resource reservations based on resource usage prediction. The model includes two metrics, namely Fail and Waste, which evaluate the quality of a reservation strategy.The effectiveness of our model is illustrated by a set of simple prediction strategies as a function of two parameters, utilization state size and the state change in subsequent reservation periods. The first parameter serves as the unit of reservation and the latter captures the maximum change in utilization accommodated by the reservations. Insights for making a proper choice for these parameters, in order to obtain an acceptable balance between the obtained results for the two resource prediction quality metrics, are revealed by exploring the parameter space.

Our future research involves more advanced prediction strategies based on prediction algorithms with higher complexity. In particular, the presence of clustered states, with state transitions occurring mainly between clusters, gives rise to the more sophisticated concept of runtime resource usage modes [23]. Since sojourn times in modes are expected to be longer than in individual states, prediction of modes should be more accurate than prediction of states, and therefore, yield improved resource management with respect to Fail and Waste.

TABLE V. FAIL AND WASTE FOR VARIOUS STRATEGIES – MOTION DETECTION COMPONENT; P_{SMP} = 780MS; N=5000

k	S(%)	2	4	5	10
0	f_k (%)	42.10	35.76	33.60	25.42
	W_k (%)	2.76	3.34	3.94	5.70
	MW(%)	0.00	0.34	0.77	2.13
1	f_k (%)	30.28	17.08	13.84	3.34
	W_k (%)	3.93	6.13	7.70	14.34
	MW(%)	0.00	0.71	1.40	3.86
2	f_k (%)	20.84	7.82	5.08	0.48
	W_k (%)	5.33	9.57	12.24	24.17
	MW(%)	0.00	0.90	1.75	4.11
3	f_k (%)	13.82	3.64	1.72	0.08
	W_k (%)	6.92	13.30	17.06	34.13
	MW(%)	0.00	0.98	1.89	4.15
4	f_k (%)	9.14	1.54	0.70	0.02
	W_k (%)	8.65	17.18	21.99	44.06
	MW(%)	0.00	1.04	1.93	4.15

REFERENCES

[1] V. Baiceanu, C. Cowan, D. Mcnamee, C. Pu, and J. Walpole, "Multimedia applications require adaptive cpu scheduling", In Proceedings of the IEEE RTSS Workshop on Resource Allocation Problems in Multimedia Systems, 1996.

[2] ViCoMo Website. [ONLINE]. Available: http:// www.vicomo.org/.

[3] I. David, B. Orlic, R. H. Mak, and J.J. Lukkien, "Towards resource-aware runtime reconûgurable component-based systems", In Proceedings of the 2010 6th World Congress on Services, SERVICES '10, pp. 465–466, Washington, DC, USA, 2010. IEEE Computer Society.

[4] J. Regehr and J.A. Stankovic, "Augmented cpu reservations: Towards predictable execution on general-purpose operating system", In IEEE Real Time Technology and Applications Symposium, pp. 141–148. IEEE Computer Society, 2001.

[5] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey", IEEE Trans. Software Eng., pp. 295–310, 2004.

[6] D. G. Sachs and D. L. Jones, "Stochastic resource allocation for energy-constrained systems", EURASIP J. Wireless Comm. and Networking, 2009.

[7] V. Vardhan, W. Yuan, A. F. Harris, S. V. Adve, R. Kravets, K.Nahrstedt, D.G. Sachs, and D. L. Jones, "Grace-2: integrating

DOI: 01.IJIT.3.1.1140

ûne-grained application adaptation with global adaptation for saving energy", IJES, pp.152–169, 2009.

[8] R. Albers, E. Suijs, and P.H.N. de With, "Resource prediction and quality control for parallel execution of heterogeneous medical imaging tasks", ICIP, pp. 2317–2320. IEEE, 2009.

[9] R. Albers, E. Suijs, and P.H.N. de With, "Resource usage prediction for groups of dynamic image-processing tasks using markov modeling", ICASSP, pp. 1929–1932. IEEE, 2009.

[10] R. Albers, E. Suijs, and P.H.N. de With, "Triple-c: Resource-usage prediction for semi-automatic parallelization of groups of dynamic image-processing tasks", IPDPS, pp. 1–8. IEEE, 2009.

[11] C. C. Wust and W. F. J. Verhaegh, "Quality control for scalable media processing applications", J. Scheduling, pp.105–117, 2004.

[12] J. Zhang, M.S. Yousif, R. Carpenter, and R.J. O. Figueiredo, "Application resource demand phase analysis and prediction in support of dynamic resource provisioning", In ICAC, pp. 12. IEEE Computer Society, 2007.

[13] E. Bondarev, M.R.V. Chaudron, and P.H.N. de With, "Compositional performance analysis of component-based systems on heterogeneous multiprocessor platforms", In EUROMICRO-SEAA, pp. 81–91. IEEE, 2006.

[14] E. Bondarev, P.H.N. de With, M.R.V. Chaudron, and J. Muskens, "Modelling of input-parameter dependency for performance predictions of component-based embedded systems", In EUROMICRO-SEAA, pp. 36–43. IEEE Computer Society, 2005.

[15] E. Bondarev, J. Muskens, P. H. N. de With, M.R.V. Chaudron, and J. Lukkien, "Predicting real-time properties of componentassemblies: A scenario-simulation approach", In EUROMICRO, pp. 40–47. IEEE Computer Society, 2004.

[16] H. Koziolek, S. Becker, and J. Happe, "Predicting the performance of component-based software architectures with different usage profiles", QoSA, vol. 4880 of Lecture Notes in Computer Science, pp. 145–163. Springer, 2007.

[17] H. Koziolek, J. Happe, and S. Becker, "Parameter dependent performance specifications of software components", QoSA, vol. 4214 of Lecture Notes in Computer Science, pp. 163–179. Springer, 2006.

[18] S. Becker, H. Koziolek, and R. Reussner, "Model-based performance prediction with the palladio component model", WOSP, pp. 54–65. ACM, 2007.

[19] M. Roitzsch and M. Pohlack, "Video quality and system resources: Scheduling two opponents", J.Visual Communication and Image Representation, pp. 473–488, 2008.

[20] M.M.H.P. van den Heuvel, R.J. Bril, S. Schiemenz, and C. Hentschel, "Dynamic resource allocation for real-time priority processing applications", In IEEE Transactions on Consumer Electronics (TCE), vol. 56, pp. 879–887, May 2010.

[21] OpenCV framework website. [ONLINE]. Available: http://opencv.willow.garage.com/.

[22] Intel website. [ONLINE]. Available: http://ark.intel.com/products/41316.

[23] I. David, M.M.H.P. van den Heuvel, R.H. Mak, and J.J. Lukkien, "Resource-usage modes detection for run-time resource prediction of video components", International Conference on Consumer Electronics (ICCE) 2013, in press.

DOI: 01.IJIT.3.1.1140